Apache Airavata Security Manager: Authentication and Authorization Implementations for a Multi-Tenant eScience Framework

Supun Nakandala
Pervasive Technology Institute
Indiana University, USA
snakanda@iu.edu

Hasini Gunasinghe
Computer Science
Purdue University, USA
huralai@purdue.edu

Suresh Marru
Pervasive Technology Institute
Indiana University, USA
smarru@iu.edu

Marlon Pierce
Pervasive Technology Institute
Indiana University, USA
marpierc@iu.edu

***Abstract*—eScience middleware frameworks integrating multiple virtual organizations must incorporate comprehensive user identity and access management solutions. In this paper we examine usage patterns for these systems and map the patterns to widely used security standards and approaches. We focus on science gateways, a class of distributed system cyberinfrastructure. Science gateways are end user environments that provide access to a wide range of academic and commercial computing and storage resources for virtual organizations. Successful gateways focus on specific scientific communities and domains, but they build on many reusable features that can be provided by general purpose hosted platform services that can support multiple tenants. Providing a security framework for identity and access management for such hosted service removes the burden for each gateway to handle its user identity management and control access to its critical resources. From the resource provider's point of view, it provides a basis for more uniform accounting and auditing. Challenges arise from the range of gateways (both legacy and newly created), the range of technologies used to build them, and the range of end user environments (Web, mobile, desktop, and programmatic API clients) that gateways provide. Using Apache Airavata as an implementation, we examine three common gateway types based on where the user identity information is held and how these can be treated in a unified manner using OAuth2 and OpenID-Connect. Our solutions for identity and access management are not specific to Apache Airavata but can be generally applied to any e-Science platform.**

***Index Terms*—science gateways, identity management, distributed systems security**

I. Introduction

Science Gateways [1] [2] [3] [4] are user environments and supporting services that help researchers make effective and enhanced use of a diverse set of computing, storage, and related resources. Gateways serve as Virtual Organizations, brokering access for their users to a broad collection of resources from different, often unaffiliated resource providers including campus grids, national scale cyberinfrastructure, and commercial cloud vendors. Thus, identity management, authentication, and authorization are critical capabilities that a gateway must provide.

Security management is one example of the general-purpose features that underlie many domain specific gateways. An important architectural trend is for gateways serving specific Virtual Organizations to use hosted, general purpose platform services. The goal of the Apache Airavata project [5] [6] is to implement and integrate many of these general purpose services into a common framework that can be run as a multi-tenanted platform service that can support multiple gateways simultaneously. In Apache Airavata, these services are implemented by multiple components (see [5]) but are exposed through a common API and a single, logical connection point, the API Server. Here, we describe the design and implementation needed to secure the interactions between a gateway tenant and the Apache Airavata API Server.

Science gateway tenants to Apache Airavata services run remotely and are typically under the control of the gateway provider. Gateway clients interact with Apache Airavata through an Apache Thrift-based API. For an overview of the API, see [6]. The Thrift-defined API and data models allow Airavata to provide client SDKs in many programming languages, including Java, PHP, Python and C++. These SDKs manage over-the-wire communications. Our technical challenge is to implement security features over the top of these communications. We note an additional challenge: gateways are not only Web-based but may also be desktop or native mobile applications. They may also be embeddable clients intended to be called directly from end-user scripts and notebooks. This introduces a challenge since Airavata SDKs are distributed directly to the end user's device. We cannot assume access is restricted behind a Web server.

In this paper, we discuss the design and implementation of the solution for securing the Airavata API. This includes authenticating and authorizing end users into the Airavata API, based on different use cases that depend on the different types of user identity management scenarios in gateways. The primary contributions of this paper are identifying general patterns for gateway identity management, mapping these patterns to OAuth2 scenarios, and implementing these solutions. We examine the implementation using three Apache Airavata gateway clients.

We first identify three scenarios in which a gateway tenant interacting with Airavata middleware may manage its users.

- Scenario 1: The gateway client does not have a user

store and will use Airavata infrastructure to provide user management.

- Scenario 2: The gateway has a user store and in-house identity management mechanisms. In this scenario, different gateways have different preferences on the level at which they share user identity information. This is typical of mature gateways.
- Scenario 3: The gateway authenticates users into the gateway using a federated identity provider such as InCommon using mechanisms such as SAML SSO, OpenID, and OAuth.

We must be able to provide a unified identity management solution that can meet the requirements of the above scenarios and provide proper Airavata API security that can be seamlessly adopted by all types of gateways including web based and native (desktop and mobile) clients.

II. SOLUTION OVERVIEW

When devising a solution, one option that we considered was to provide user authentication at the gateway layer using the authentication protocol of the gateway's choice, and use the system-to-system authentication mechanism between the gateway and the Airavata server. Examples of two of the system-to-system authentication mechanisms that can be used are mutual authentication using SSL certificates and basic authentication (i.e: Gateway credentials over TLS). The main drawback of the certificate-based mutual authentication approach is that it is not scalable from the management point of view. Each gateway can have different types of applications (web, native), and ideally each of these applications should have different credentials. When the number of gateway clients increases, management of gateway credentials and PKI infrastructure is not scalable [7]. Also, both of these approaches can be used only for web based gateways, where credentials or certificates can be securely maintained in a web server. They are not secure enough to be used in native clients as a malicious user can reverse engineer the client and gain access to the credentials/keys. One way to facilitate native clients using these approaches is to route the requests from native clients via a proxy server and establish the mutual trust between the proxy server and Airavata using gateway credentials or certificates. This requires extra effort from gateway developers and requires end user information to be passed to the Airavata API Server by the clients with every API request.

The solution we adopted is to use OAuth 2.0 based authorization delegation [8] to the gateway by the user authenticated at the gateway. The main advantage of this approach is scalability. This approach does not require any management of gateway credentials or PKI infrastructure. The OAuth access tokens can be generated by a separate dedicated authorization server and not by the API Server. This approach also benefits from wider adoption in the general "Software as a Service" and "Platform as a Service" communities to which science gateways and gateway frameworks belong. OAuth 2.0 is the de-facto standard for access delegation. It can be used in conjunction with existing authentication protocols such as SAML 2.0 based single sign on via the OAuth 2.0 extension profiles. Even though OAuth is an authorization delegation protocol, authentication of users can be done using OpenID-Connect [9]. Hence OpenID-Connect + OAuth 2.0 can be used as a comprehensive authentication and authorization protocol.

Figure 1 illustrates the high level architecture of the solution with mappings to the standard OAuth 2.0 based authorization-delegation solution architecture. Because OAuth 2.0 and related technologies are widely adopted, we chose to integrate a third party implementation rather than develop it ourselves. Our solution makes use of the identity management features offered by WSO2 Identity Server (IS) [10]. WSO2 IS is a widely used open source (Apache V2 License) identity management system that provides out-of-the-box support for many identity management standards such as OAuth/OpenID, SAML SSO, XACML, and features such as multi-factor authentication, password policies, etc. It supports multi-tenancy, per-tenant user store configuration, and custom pluggable user store manager extensions, all of which are relevant to the problems discussed in this paper.

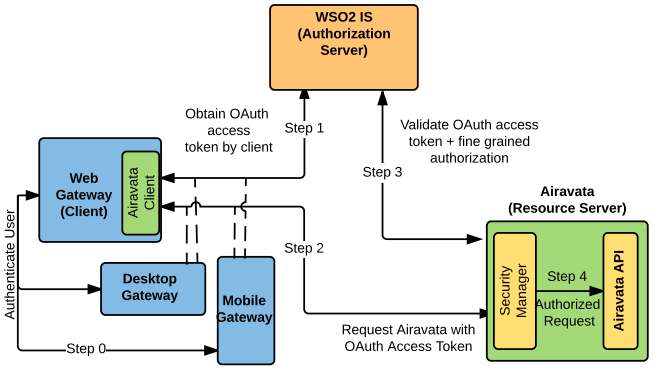

Fig. 1. High level overview of the solution

Details of the interactions illustrated in Figure 1 are as follows.

0) User is authenticated to Airavata. This paper examines three specific scenarios.
1) OAuth token is obtained from the WSO2 IS to access the Airavata API on behalf of the authenticated user.
2) Requests to Airavata, depending on the actions the user wants to perform, are sent along with the obtained OAuth token.
3) Each request sent to the Airavata API Server is authorized by the Security Manager. The Security Manager acts as a client within Airavata to various WSO2 IS and other potentially services. First the attached OAuth token is validated and user's profile information is retrieved. Then the request is authorized; authorization approaches include role-based or group based decisions.
4) Only the authorized requests are allowed to reach the Airavata API.

Authorization decisions for subsequent requests from the same user are handled using caching to avoid contacting WSO2

IS for every call. This step can use different authorization mechanisms, such as role determination using XACML capabilities of IS or group membership via Grouper. The above solution supports multi-tenancy; when the user identity is exchanged, it includes the tenant domain that the user belongs to so that the authentication and authorization are performed with respect to that tenant domain in WSO2 IS. The same client side logic to authenticate a user and obtain an OAuth token can be implemented in web, desktop and mobile clients using the recommended grant types available in the OAuth 2.0 specification.

III. SOLUTION IN DETAIL

A. Authenticating & Obtaining an OAuth 2.0 Access Token

User authentication and obtaining an access token by the client application (Step 0 and Step 1 in Figure 1) are the only steps in the high level solution that differ among the three scenarios that we identified. OAuth 2.0 specifies five grant types to obtain an access token by the client app. Each of these grant types serves a different purpose and is used in a different way. We evaluate each and discuss their applicability to our gateway client scenarios.

The Authorization Code grant type is the recommended grant type to be used if the client application is capable of spawning a web browser to redirect the user to the authorization server's authentication page. The user signs in to the authorization server and authorizes the gateway application to obtain an access token for the user. This authorization is a one-time-only application authorization. If the authorization server authorizes the request, the user will be redirected back to the client with a token (called the authorization code) in the query string, which the client application will capture and exchange for an access token in the background. In order to use this grant type, the application itself should already be trusted by the authorization server through some application registration process. This is applicable for standard Web-based gateway tenants.

The Implicit grant type is very similar to the Authentication Code grant type. The difference is that the access token is directly returned to the client application after the user signs in for Implicit grants. Implicit grants are for clients that are not capable of keeping the client application's own credentials secret. An example scenario is a thick browser client that uses JavaScript to directly access the Apache Airavata API methods rather than going through an intermediate Web server.

If the user (resource owner) already trusts the client application to provide his/her own credentials, the Authorization Code grant type is not needed. Resource Owner Password grant type is an alternative that can be used to simplify the flow in this case. This grant type is useful in scenarios where the client application cannot spawn a web browser; examples include desktop and mobile client applications. In this grant type instead of redirecting the user to the authorization server's authentication page, the client application itself obtains the user credentials and then sends these to the authorization server along with the client's own credentials. If the authentication is successful then the client will be issued an access token.

The Client Credential grant type is similar to the Resource Owner Password grant except the client application's credentials are used to authenticate a request for an access token. This grant type should only be used by trusted clients. This grant is suitable for machine-to-machine authentication that involves no user interaction or user authorization. In this grant type we do not need to use OpenID-Connect to authenticate users.

Authorization servers that support Refresh Code grant types are able to issue a refresh token when they return an access token to a client application. When the access token expires, the client can use the refresh token to retrieve a new access token with the same permissions. The client has to maintain the state of each token. This can be done by either periodically using the active refresh token or by using the refresh token to acquire a new token after an access failure.

B. Science Gateway Client Scenarios

We now map the general OAuth2 solutions to three different gateway identity management scenarios. All solutions use the Security Manager component within Apache Airavata (see Figure 1) that acts as a client to various WSO2 IS services.

1) The gateway doesn't have a user store and would like to depend on Airavata to provide user management features: The gateway application makes use of the Airavata Client SDK to create users, which in turn invokes the User Admin API of the WSO2 IS; see Figure 2. When users are authenticated to the gateway, their credentials are validated against those stored in the WSO2 IS user store. The gateway uses Airavata's client SDK to authenticate users and obtain OAuth access tokens using OpenID-Connect. All the client applications are provided by the gateway itself and no user generated or third party applications need to connect to Airavata. Thus all gateway client applications can be considered as trusted clients, and therefore instead of Authorization Code grant type we use the Resource Owner Password grant type to obtain access tokens for both web and native clients. If the web application is a browser application that directly interacts with the Airavata API through a JavaScript SDK, the Implicit grant type should be used instead and privileges of access token obtained by this grant type should be highly restricted.

2) The gateway has its own user-store and identity management mechanisms: Here we consider three solution categories to address this scenario based on the differences in preference of the gateway to share user identity information with Airavata. In the first category, the gateway will not share any information about the end user's identity and will use Airavata only as a job execution engine. Therefore the gateway client will obtain an OAuth access token by authenticating to the WSO2 IS. The Client Credential grant type is the appropriate type for this case. After retrieving an access token, end user requests to Airavata attach this token (see Figure 3). In this case the types of gateway applications that can be supported are limited to web based gateways where the client credentials can be securely maintained in the web server. For native (JavaScript

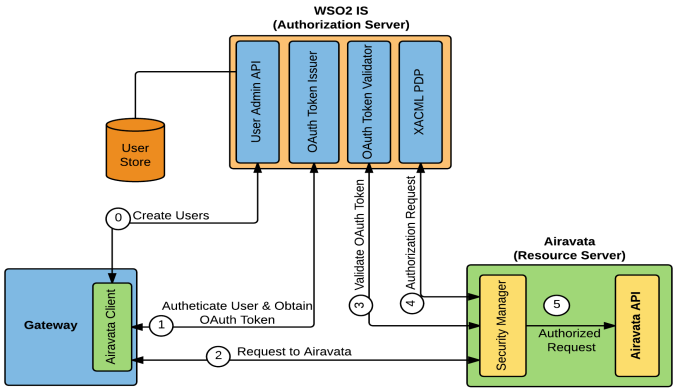

Fig. 2. Gateway uses Airavata-provided user store

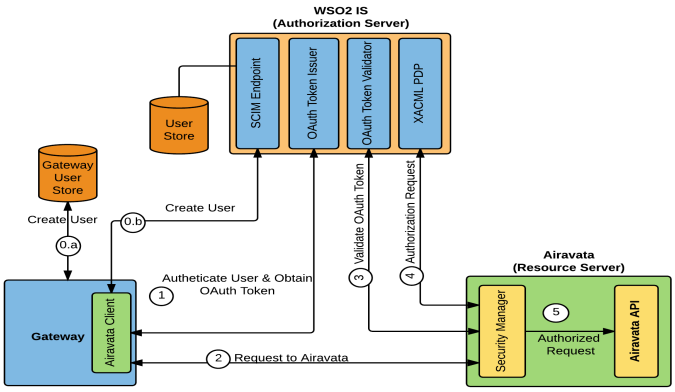

Fig. 4. Gateway provisions user accounts to the Airavata provided user store

and desktop) client gateways that fall in this category, the requests should be sent through an intermediary proxy server that can securely maintain the client credentials.

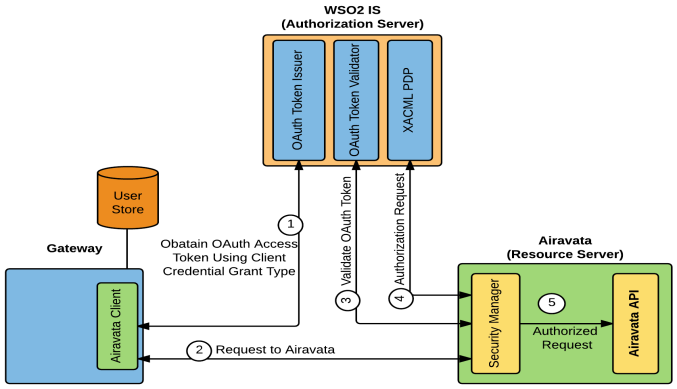

Fig. 3. Gateway does not share any user identity information with Airavata

In the second category, the gateway shares user identity information but does not allow Airavata to connect to the gateway's user store. User accounts need to be provisioned with the identity information required by Airavata. In this case the gateway uses Airavata's identity provisioning SDK client to provision user accounts to Airavata's identity manager (i.e WSO2 IS) at the time of user creation in the gateway. Already created user accounts in the gateway user store can be provisioned through a bootstrapping phase. Invoking the Airavata SDK for user provisioning will in turn invoke the System for Cross-domain Identity Management (SCIM) [11] endpoint in WSO2 IS for identity provisioning. This enables the execution flow of accessing the Airavata API to be the same as in Scenario 1; see Figure 4. In this category user information will be duplicated in both the gateway user store and the WSO2 IS user store. It is the responsibility of the gateway to maintain them in coherence.

In the third category, the gateway allows Airavata to connect to its organizational user store in read-only mode. In this case, the identity manager of Airavata (i.e WSO2 IS), is connected to the gateway's organizational user store through the user store manager extension provided by the WSO2 IS. This enables the user authentication, access token retrieval, and the rest of the execution flow for accessing the Airavata API to be the same as in scenario 1; see Figure 5. The WSO2 IS distribution by default provides user store manager extensions that can be used to integrate Active Directory and LDAP based user stores. Custom user store manager extensions can be written to integrate any custom user store, such as a database.

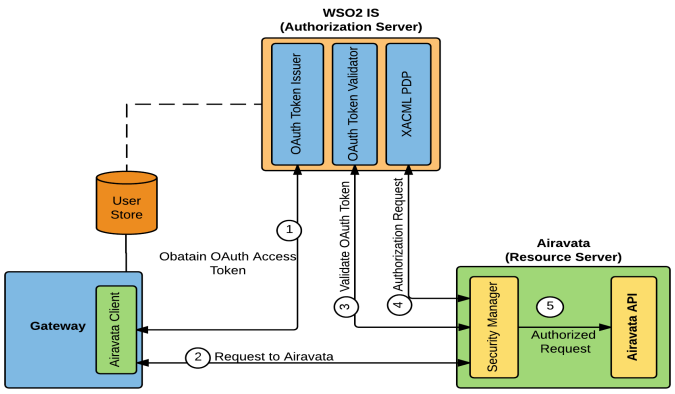

Fig. 5. Gateway allows Airavata to have read only access to the organizational user store

3) The science gateway does not have a user store but instead authenticates users into the gateway using a third party federated identity provider: In this scenario the gateway becomes a relying party, and it needs to authenticate users to Airavata through the federated authentication mechanism that is being used in the gateway community. To solve this issue, we used the inbound authentication configuration feature in WSO2 IS. Using this feature, it is possible to plug external federated authentication providers as inbound authentication providers to a user store. Out of the box, WSO2 IS supports OpenID, SAML, OAuth2/OpenID-Connect, WS Federation, Google, Yahoo, Microsoft and Facebook federated authentication providers. For a federated authentication protocol not in the previous list, it is possible to plug in a custom inbound authentication provider.

If the federated authentication protocol supports retrieval of the user's identity attributes from the identity provider (such as

SAML, OpenID), a user account is created in WSO2 IS with this identity information using just-in-time provisioning. When a user tries to authenticate to WSO2 IS, the user can select the configured identity provider and login to that provider. Once the user is authenticated via the federated identity management protocol, an OAuth access token is generated in WSO2 IS that has access to the Airavata API, and the rest of the flow of execution continues in the same way as in other cases; see Figure 6. Since federated authenticators typically support only web-based access, a web based OAuth 2.0 grant type such as Authorization Code grant is appropriate. In the case of thick web clients, Implicit grant type should be used. Native clients are now required to spawn an embedded browser for user authentication and extract the access token.

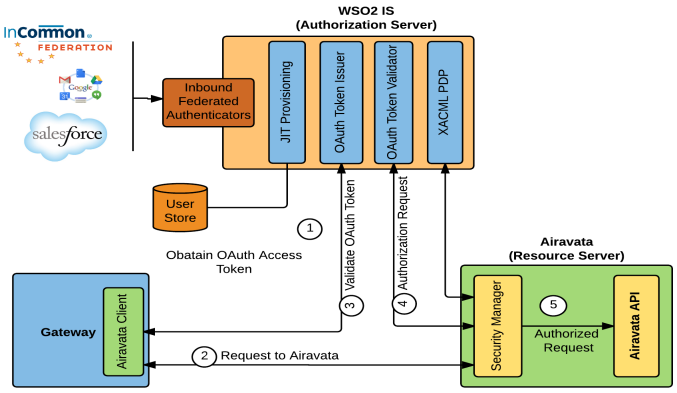

Fig. 6. Gateway uses federated identity provider.

IV. IMPLEMENTATION AND EVALUATION

Every Apache Airavata API method definition was changed to incorporate a mandatory field named AuthzToken, which contains the OAuth access token and the Airavata tenant ID information that is set at the gateway client layer. To process this token, we added the new Security Manager component to the Airavata API Server. This new component is designed to be pluggable so that we can change it if required without affecting the rest of Airavata. The Security Manager intercepts all API requests and validates them from the separately hosted WSO2 Identity Server, which acts as the Authorization Server for Airavata. For intercepting the API requests the Security Manager uses annotations supported by the Google Guice library.

Authorization validation at the Security Manager is a two step process. First the OAuth access token is validated by calling the OAuth token validator endpoint in WSO2 IS, and then any authorization policy is enforced. When invoking these endpoints, the requests need to be authenticated using HTTP basic authentication by providing the respective tenant's admin credentials in WSO2 IS. Therefore, Airavata needs to know and store the respective credentials for IS tenants. For this we used Airavata's credential store server [12]. Gateway administrators can store their IS tenant credentials by updating the GatewayProfile object in Airavata.

Ideally, every Airavata API request should be validated against the WSO2 IS as shown in Step 3 in Figure 1. However, we found that this imposes a significant performance overhead (about 350 ms) on the Airavata API method latency. To overcome this issue we added an authorization cache module into the Security Manager that caches the authorization decisions for each access token-tenant-API method combination after successful validation from WSO2 IS. Subsequent API requests to the API Server from the same user are checked in the authorization cache instead of directly invoking the IS validation endpoints. The caching duration of each authorization decision is set to the remaining valid duration of the corresponding OAuth access token, which can be obtained from the OAuth token validation response. This OAuth token validation response contains various user attributes such as username and email that are also cached.

We have implemented the designs described here to support several science gateway tenants to Apache Airavata. The SEAGrid science gateway [13] is an example of a Scenario 1 tenant. SEAGrid uses a MySQL database as the user store in WSO2 IS and provides a web based gateway and a desktop client. Both of these clients use the Resource Owner Password grant type in OAuth 2.0 specification to authenticate users and retrieve access tokens as both are trusted clients. Another client is a USDA-funded science gateway, currently under development by a collaborating team, for accessing bioinformatics applications. This client needs to integrate the user management with the existing user store. Thus this falls to the Category 3 of Scenario 2 in our taxonomy. The USDA user store is an LDAP server, and hence it was straightforward to integrate with IS. This gateway is web-based, so a Resource Owner Password grant type is used for authentication and access token retrieval. The UltraScan science gateway [14] provides its own user store and does not share any user information with Airavata. Thus, it is a Category 1, Scenario 2 gateway in our taxonomy. As described in the solution section, the Client Credential grant type is used to retrieve an access token that is used to submit API requests to Airavata.

V. RELATED WORK

Science gateway security, especially the aspect of user credential management for individual gateways, is a well studied area. One approach is to support end-to-end single sign on for users. Whenever a gateway user submits a job to a remote compute resource, the gateway middleware uses that particular user's credentials on the remote host for authentication. Per user grid certificates, MyProxy and various MyProxy services such as OAuth for MyProxy [15], CILogon for MyProxy, and the MyProxy Gateway are implementations of this approach [16]. Security Assertion Markup Language based Single-Sign-On (SAML SSO) is used by the MoSGrid science [17].

The per-user credential mechanism can be difficult to operate in practice. The Authentication, Authorization, Auditing and Accounting (AAAA) model [18] is an alternative; XSEDE uses a simplified version of this model in production. The main

idea of the AAAA model is to adopt community user accounts and avoid using per user credentials to authenticate to the remote resources. This significantly reduces the effort required for user management tasks in compute resources as now there are a limited number of science gateway community accounts. This approach outsources gateway user management tasks to the gateway layer itself. In this model, multiple gateway users share the same community account to submit jobs to compute resources. Within Airavata, community credentials are managed by the credential store component [12].

Globus Nexus [19] is very similar to WSO2 IS. It is a "platform as a service" application that provides user identity management, profile management, and group management features. Its identity management capabilities allow users to create a unique Globus identity that can be associated with federated external identities from campus identity providers (e.g CILogon), computing resource identity providers (XSEDE accounts using MyProxy OAuth), and commercial identity providers such as Google. This Globus Identity can be consumed by subscribing applications using an OAuth-based workflow to create a Single-Sign-On environment. Globus Nexus is a hosted service, whereas WSO2 IS is downloadable, open source software that is also available as a for-fee service. This difference introduces tradeoffs that gateways and gateway platform service providers should consider.

VI. CONCLUSIONS AND FUTURE WORK

This paper examines the over-the-wire access patterns that exist between a wide range of gateway clients and multi-tenanted platform services like Apache Airavata and how they can be mapped to OAuth2 protocol and OpenID-Connect. This paper does not cover fine-grained authorization decisions. These can be implemented either as roles or as groups; our current implementation uses XACML role expressions within WSO2 IS to encode and enforce roles associated with different API calls. We are currently comparing this approach with group-based approaches that can be implemented using open source Grouper software.

ACKNOWLEDGMENT

This work was supported by NSF award #1339774 "Collaborative Research: SI2-SSI: Open Gateway Computing Environments Science Gateways Platform as a Service (OGCE SciGaP)". S. N. and H. G. were supported by Google Summer of Code. The Center for Trustworthy Scientific Cyberinfrastructure (NSF awards #1234408 and #1547272) consultations are summarized at http://trustedci.org/scigap/.

REFERENCES

[1] K. A. Lawrence, M. Zentner, N. Wilkins-Diehr, J. A. Wernert, M. Pierce, S. Marru, and S. Michael, "Science gateways today and tomorrow: positive perspectives of nearly 5000 members of the research community," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 16, pp. 4252–4268, 2015.

[2] S. Gesing and N. Wilkins-Diehr, "Science gateway workshops 2014 special issue conference publications," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 16, pp. 4247–4251, 2015.

[3] N. Wilkins-Diehr, S. Gesing, and T. Kiss, "Science gateway workshops 2013 special issue conference publications," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 2, pp. 253–257, 2015.

[4] P. Kacsuk, *Science Gateways for Distributed Computing Infrastructures*. Springer, 2014.

[5] S. Marru, M. Pierce, S. Pamidighantam, and C. Wimalasena, "Apache airavata as a laboratory: architecture and case study for component-based gateway middleware," in *Proceedings of the 1st Workshop on The Science of Cyberinfrastructure: Research, Experience, Applications and Models*. ACM, 2015, pp. 19–26.

[6] M. Pierce, S. Marru, B. Demeler, R. Singh, and G. Gorbet, "The apache airavata application programming interface: overview and evaluation with the ultrascan science gateway," in *Proceedings of the 9th Gateway Computing Environments Workshop*. IEEE Press, 2014, pp. 25–29.

[7] P. Gutmann, "Pki: it's not dead, just resting," *Computer*, vol. 35, no. 8, pp. 41–49, 2002.

[8] D. Hardt, "The oauth 2.0 authorization framework," 2012.

[9] N. Sakimura, J. Bradley, M. Jones, B. de Medeiros, and C. Mortimore, "Openid connect core 1.0," *The OpenID Foundation*, p. S3, 2014.

[10] W. Inc, "Wso2 identity server: The first enterprise identity bus," http://wso2.com/products/identity-server/.

[11] K. Grizzle, E. Wahlstroem, C. Mortimore, and P. Hunt, "System for cross-domain identity management: Core schema," *System*, 2015.

[12] T. A. Kanewala, S. Marru, J. Basney, and M. Pierce, "A credential store for multi-tenant science gateways," in *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*. IEEE, 2014, pp. 445–454.

[13] S. Pamidighantam, S. Nakandala, E. Abeysinghe, C. Wimalasena, S. Rathnayaka, S. Marru, and M. Pierce, "Community science exemplars in seagrid science gateway: Apache airavata based implementation of advanced infrastructure," *Procedia Computer Science*, vol. 80, pp. 1927–1939, 2016.

[14] B. Demeler and G. E. Gorbet, "Analytical ultracentrifugation data analysis with ultrascan-iii," in *Analytical Ultracentrifugation*. Springer, 2016, pp. 119–143.

[15] J. Basney, R. Dooley, J. Gaynor, S. Marru, and M. Pierce, "Distributed web security for science gateways," in *Proceedings of the 2011 ACM workshop on Gateway computing environments*. ACM, 2011, pp. 13–20.

[16] J. Basney, J. Gaynor, S. Marru, M. Pierce, T. A. Kanewala, R. Dooley, and J. Stubbs, "Integrating science gateways with xsede security: A survey of credential management approaches," in *Proceedings of the 2014 Annual Conference on Extreme Science and Engineering Discovery Environment*. ACM, 2014, p. 58.

[17] S. Gesing, R. Grunzke, J. Krüger, G. Birkenheuer, M. Wewior, P. Schäfer, B. Schuller, J. Schuster, S. Herres-Pawlis, S. Breuers *et al.*, "A single sign-on infrastructure for science gateways on a use case for structural bioinformatics," *Journal of Grid Computing*, vol. 10, no. 4, pp. 769–790, 2012.

[18] J. Basney, V. Welch, and N. Wilkins-Diehr, "Teragrid science gateway aaaa model: implementation and lessons learned," in *Proceedings of the 2010 TeraGrid Conference*. ACM, 2010, p. 2.

[19] K. Chard, M. Lidman, B. McCollam, J. Bryan, R. Ananthakrishnan, S. Tuecke, and I. Foster, "Globus nexus: A platform-as-a-service provider of research identity, profile, and group management," *Future Generation Computer Systems*, vol. 56, pp. 571–583, 2016.